Reactive Programming using Procedural Parameters for End-User Development and Operations of Robot Behavior Control

著者	Floris Marc Arden Erich
内容記述	この博士論文は内容の要約のみの公開（または一部非公開）になっています
year	2018
その他のタイトル	エンドユーザによるロボット行動制御のための手続型変数を用いたリアクティブプログラミング
学位授与大学	筑波大学 (University of Tsukuba)
学位授与年度	2017
報告番号	12102甲第8751号
URL	http://hdl.handle.net/2241/00152355

Reactive Programming using Procedural Parameters for End-User Development and Operations of Robot Behavior Control

March　２０１８

ERICH Floris Marc Arden

Reactive Programming using Procedural Parameters for End-User Development and Operations of Robot Behavior Control

School of Integrative and Global Majors
Ph.D. Program in Empowerment Informatics
University of Tsukuba

March　２０１８

ERICH Floris Marc Arden

Abstract

This thesis explores robot programming from the perspective of end-users. It takes under consideration not just the development of robot software, but also how the robot is operated. It is based on research in the fields of robotics, software engineering and human computer interaction. By combining these fields I hope to make a significant contribution to how humans interact with robots in the future.

Whereas in the past robots have been confined to factory environments, recently affordable robots such as NAO from Softbank Robotics and OP2 from ROBOTIS have become available for use in personal environments. These robots can be programmed to do various tasks desired by the user, such as performing demonstrations, communicating information retrieved from the Internet and interacting with objects in the environment. Even so, currently robots are not as ubiquitous in our environments as one might expect them to be.

Robots are deployed into an environment that they interact with through sensing and actuation. The robot senses for stimuli and actuates a response after processing the stimuli. The components of the robot that sense for stimuli are called sensors, and the components that actuate a response are called actuators. Sensors and actuators are physical components embedded inside of the robot.

To connect sensors and actuators, some processing is necessary. Because the robots that we consider in this thesis are of general purpose, the processing has to be reconfigurable, or programmable, using software. Hence the sensors and actuators of the robot have to be exposed within a software platform, and some means of programming has to be offered to reconfigure the processing done by the robot.

An example of a field in which such a platform might be used is Socially Assistive Robotics (SAR). In SAR, robots use their sensors and actuators to exhibit behaviors that model human social behaviors. Researchers hypothesize that using a robot in this way allows them to have great precision in teaching certain social behaviors [87]. Children also tend to react favorably to the toy-like appearance of the robot platforms considered in this thesis. Hence, therapists are interested in using robots as a therapeuthic instrument.

Currently a SAR experiment requires the following participating roles: A developer (either a person or a group of people), an operator (typically a single person) and a user (the therapist). Before the experiment, the therapist will communicate with the developer to design software that the therapist can use in the experiment. During the experiment itself there is also an operator present who controls the robot through the software written by the developer.

The need for communication between these three roles creates a long lead time before experiments can be performed. Details can get lost during communication,

which can cause bugs in the software. Ideally, the therapist would be able to develop and operate the robot software with minimal support from other people and groups. The therapist should in this case take on the roles of developer and operator.

Robot programming is however notoriously complex. Robots are distributed embedded systems operating in physical environments. Writing software for robots requires knowledge of physics, concurrency, computational constraints, how to deal with dynamic environments and how to ensure safety of both the user and the robot.

In Software Engineering, the problem of interaction between development and operations has recently gathered a lot of interest both in academic and in professional circles. Many organizations have adopted an approach called DevOps to try to improve the interaction between development and operations. DevOps is formally defined as a set of practices that tries to reduce the time required for changes in a system to evolve from development to operations, without negatively affecting quality [11]. In practice however organizations define DevOps as simply the interaction between development and operations [42].

In this research I aim to create a programming environment that allows end-users to develop and operate applications in Socially Assistive Robotics experiments. To that end I will answer the following research questions:

- What applications are used in SAR experiments?
- What other approaches exist to develop and operate robots?
- What are the components of the programming environment?
- How effective is the programming environment?
- How can the programming environment be used in different contexts?

To answer the question of what applications are used in SAR experiments, I performed a systematic literature review. Our search returned 24 papers, from which 16 were included for closer analysis. To do this analysis I used a conceptual framework inspired by Behavior-based Robotics. I was interested in finding out which robot was used (most use the robot NAO), what the goals of the application were (teaching, assisting, playing, instructing), how the robot was controlled (manually in most of the experiments), what kind of behaviors the robot exhibited (reacting to touch, pointing at body parts, singing a song, dancing, among others), what kind of actuators the robot used (always motors, sometimes speakers, hardly ever any other type of actuator) and what kind of sensors the robot used (in many studies the robot did not use any sensors at all, in others the robot frequently used camera and/or microphone). The results of this study can be used for designing software frameworks targeting Humanoid Socially Assistive Robotics.

Existing platforms exist that allow robots to be programmed. Major examples of this are the Robot Operating System (ROS), Choregraphe and Targets-Drives-Means (TDM). ROS is aimed towards robotics researchers and not end-users. Choregraphe is not explicitly aimed towards end-users, but aims to be a beginner friendly programming environment. TDM is aimed towards end-users, and a user-friendly programming interface for TDM has been developed for the Android smartphone operating system.

In ROS an application consists of nodes that are connected to other nodes through topics. Nodes can subscribe to topics, after which they will get notified when a node publishes to this topic. The structure created through this mechanism can be described as a graph in which the nodes and topics are vertices connected by edges signalling publishing and subscribing.

In Choregraphe an application consists of boxes that are connected to other boxes through ports. Boxes can have ports to start and stop the box, as well as data input and output ports. Boxes act like small machines that accept inputs, can do some processing and can produce output. Custom boxes can be build, but, as is shown in this thesis, even expert users of Choregraphe encounter difficulty in that task. It is especially hard to create boxes that combine different types of data.

Target-Drives-Means is a robot software architecture focused on end-user development, and it is based on the behavior-based robotics paradigm [7]. In TDM a program is a collection of behaviors. A behavior is a collection of action groups. A complex behaviour is a behavior that includes other behaviors. For complex behaviors the action groups of the included behaviors are joined to form the action groups of the complex behavior. An action group is a collection of action units. An action unit is composed of an action and a condition.

Platforms such as ROS, Choregraphe and TDM need facilities that go beyond connecting nodes (ROS), boxes (Choregraphe) or actions/conditions (TDM) in order to make them powerful enough for an end-user to develop new applications with. Users need to be able to develop custom nodes (ROS), boxes (Choregraphe) and actions/conditions (TDM). Currently the users can use the following for doing this:

- ROS: Developing nodes using supported programming languages such as C++ and Python.
- Choregraphe: Developing boxes using Python, subgraphs and specialized builders for movement and dialog.
- TDM: Developing actions and conditions using Python.

Both ROS and TDM require the user to have a good knowledge of programming to be a proficient user of the platforms. Choregraphe has more powerful facilities for end-users, however if a user wants to create behaviors, outside of movement and dialog, some experience of programming is necessary. I believe that an easier method for end-users to develop applications for robots exists. In this thesis I will present this and compare it with Choregraphe to show that this new method is indeed beneficial.

I call this new method Reactive Robot Programming (RRP). It is based on recent developments in software engineering, were programming languages are being extended with the capability to process streams of data [71]. Low level robot behaviors can be separated into three tasks: Sensing, Computation/Planning and Acting [7]. In RRP sensors and actuators can be connected using connectors and intermediary streams. The combination of sensors, streams, actuators and their connectors forms a graphical structure that I call the RRP Graph.

RRP is more usable than other solutions thanks to the following mechanisms:

- Streams are a natural way of reasoning about events happening in the real world.

- Seperating sensing, planning and acting makes it easy to reason about an RRP Graph.
- Having a small set of connectors to learn makes it easy to get started with using RRP.
- People are naturally visually oriented, a visual notation leverages this innate ability.

An important aspect of the RRP Graph is the usage of procedural parameters. In the RRP Graph, most connectors accept a procedure as a parameter. Practically, the connector in this case specifies what should be done, while the procedural parameter specifies how to do it. The following connectors are currently supported:

map Uses a procedural parameter to map inputs to outputs. Useful for example for doing a calculation on the input (e.g. a coordinate transform or distance calculation) or selecting a specific value from a structure (e.g. taking the value from a field of an object or looking up an index in an area).

filter Uses a procedural parameter to filter inputs that do not match a certain predicate. Useful for implementing range filters (e.g. filtering objects that are close or far) and category filters (e.g. filtering objects that have been seen before or are new; filtering faces that have been recognized or have not been recognized).

timestamp Adds a timestamp to the inputs. Useful for reasoning based on time (e.g. when combining two streams, ensure a maximum time difference between two events).

sample Samples an input at a certain rate. Useful for reducing unneeded computation and avoiding overloading an actuator.

combineLatest Uses a procedural parameter to combine inputs from multiple streams. Useful for sensor fusion.

merge Merges inputs from multiple streams. Useful when multiple streams produce the same type of output (e.g. when having redundant sensors such as two sonar sensors).

An interpreter called the RRP Runtime can read the RRP Graph and based on it initialize the sensors and actuators on the robot. The RRP Runtime will also ensure that data will flow from sensors to actuators through connectors as specified by the RRP Graph. As part of this the RRP Runtime removes the need for the user to take into consideration concurrency inside the software domain, however the user still has to consider concurrency in the physical domain (i.e. conflicts arising because certain sensors and actuators cannot be active at the same time due to logical constraints, for example an arm can only be in one position at the same time).

A Visual Programming Environment (VPE) enables the specification of programs using the RRP Graph by end-users. Procedural parameters can either be specified directly as a body of connectors, or can be stored as a helper in which case they can be reused by multiple connectors. The VPE is implemented as a web

application to allow it to be used by multiple devices without needing seperate native applications. The VPE is a Single Page Application (SPA) and hence does not require reloading the page at any time during its usage. This is realized by using Web Sockets that maintain a client-server connection while the VPE is being used. Additionally the VPE has support for multiple users interacting simultaneously.

The VPE uses a graph database for storing the applications created by users. A graph database naturally supports the structure of the RRP Graph. Inside the graph sensors, actuators and streams are stored as nodes, while simple connectors are stored as edges. Connectors are considered simple if they connect a single input stream to a single output stream and either do not take a procedural parameter or define the procedural parameter directly as a body of the connector (i.e. they do not use a helper). In the other cases the connector will be stored as a node instead.

The VPE and the Runtime make use of various layers of abstractions to make it easier to add support for different databases (currently only the Neo4j graph database is supported, but support for XML files would be a valuable addition), robots (currently only Softbank NAO is supported) and even execution models (for example using either multithreading, multiprocessing or even grid computing) in the future.

Validation of the programming environment is done through various case studies and by performing a user study. In the case studies I show how the RRP Graphs can be used to construct various useful behaviours for a robot. In one such application the robot changes its tracking behavior of an object based on the distance to the object. I then show that this behavior can be ran on a robot [40]. Each case study is not only designed as RRP Graph but also implemented using the VPE. In the user study I show that end-users, people with little programming experience, can use the VPE to easily explain, debug and create behaviours. I also show that based on the time tasks took to complete in our VPE and a state of the art commercial platform (Choregraphe), our VPE is competitive with Choregraphe in the explanation and debugging task and exceeds the capabilities of Choregraphe in the creation task [44]. Based on self evaluation by the participants using NASA-TLX, our VPE and Choregraphe are competitive for the explanation and debugging tasks, while our VPE again exceeds the capabilities of Choregraphe in the creation task.